TL;DR: Improving Abstractive Summarization Using LSTMs

Sam Kim Sang Goo Kang

Department of Computer Science, Stanford University

Mentor: Abigail See

Abstract

Traditionally, summarization has been approached through *extractive* methods. However, they have produced limited results. More recently, neural sequence-to-sequence models for *abstractive* text summarization have shown more promise, although the task still proves to be challenging. In this paper, we explore current state-of-the-art architectures and reimplement them from scratch. We begin with a basic sequence-to-sequence model. We upgrade the model to use our own attention mechanism and change the encoder to a bi-directional LSTM. Finally, we introduce and implement a hybrid pointer-generator model that points back to words in the source text if it encounters an out-of-vocabulary (OOV) word. Our qualitative results show our models successfully working by producing very similar results to the ground truth summaries. The pointer-generator model, it correctly replaces OOV words with key words from the source, particularly for numbers. Our models also do not experiences short-comings from other models such as repeating summaries.

Introduction

Humans are very capable of summarizing articles in a couple of sentences that serve well for titles and headlines. Since this involves the ability to capture the core meaning of the input text, summarization has been a tremendous challenge in the field of natural language processing. Traditionally, successful approaches have mostly been extractive, which identify key words or phrases in the text and combine them together to form a summary. More recently, researchers have been taken more of an abstractive approach, which is a bottom-up method that captures semantics that are not necessarily in the original text.

To improve upon recently successful abstractive methods, we plan to take advantage of the sequence-to-sequence model and use an LSTM for the encoder and decoder.

In this paper, we describe our models and discuss its implementation. We discuss the advantage of our last pointer-generator model to deal with OOV words that appear in the source text. For generating summaries, we describe how our two methods for decoding: greedy search and beam-search. Lastly, we highlight important results and qualitative findings from our models, showing very promising example summaries from the test set, but also describe some of their limitations.

Background

To define the problem, we want to map an input sequence of M word vectors, $x_1, \ldots, x_m$ from a fixed vocabulary V to an output of another sequence of length $N < M$ word vectors, also from the same vocabulary V but necessarily from the input sequence. Unlike some other models, we won't assume N is fixed to allow for different length summarizations. We seek to optimize

$$\underset{y \in Y}{\operatorname{argmax}}\ s(x, y) \tag{1}$$

where s is some scoring function and y is the total vocabulary. This differs from extractive models that optimize

$$\underset{m \in \{1,\ldots,M\}^N}{\operatorname{argmax}}\ s(x, x_{[m_1,\ldots,m_N]}) \tag{2}$$

which extracts words from the input article to use as the summary.

Related Work

Many previous work have implemented various summarization using other algorithms. A team

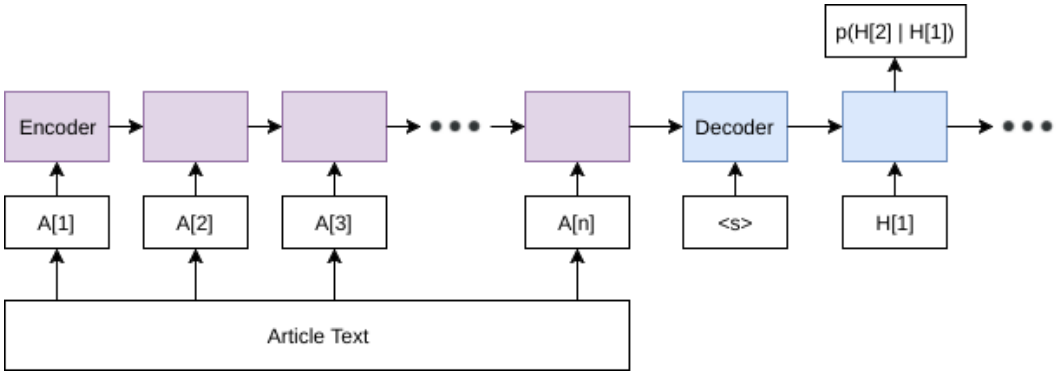

Figure 1: Basic Seq2seq model

in UC Berkeley was able to use a compression algorithm to extractively find the best combination of phrases in a piece of text that represents the whole article, using grammatical constraints to ensure a grammatically correct summary. In this method, various compression algorithms were used to try to achieve the best summarization[3].

LSTMs are very powerful models for understanding large datasets such as text, and being able to understand what it means. Due to this, it has been shown to be a very powerful model in summarization. A publication in ECCV 2016 shows how LSTM can be used to summarize videos by attempting to locate the frames of the video that best represent the whole video [4].

In Nallapati et. al (2016) [6], they approach this summarization task using sequence-to-sequence RNNs with several optimizations to deal with out-of-vocabulary (OOV) tokens. They introduce a pointer model that uses a 'switch' per time-step that determines whether to use a word from the vocab or to point back to a particular word in the input. However, it does not take advantage of the log probabilities produced by copying a word from the input *and* and generating it from the fixed vocabulary.

See et. al (2017), [8] fixes this problem by creating an extended distribution that denotes the union of the vocabulary and all of the words appearing in the source document, thus taking advantage of the attention distribution as well as softmax probabilities from the vocab. We follow this model to deal with OOV words that appear in our dataset.

Models

In this section, we describe our (1) basic sequence-to-sequence encoder decoder model as our baseline, (2) attention model, (3) bi-directional LSTM model for the encoder, and finally (4) our pointer-generator model. All models were not made from any built-in Tensorflow libraries. They were all written from scratch, besides the LSTM cell.

Sequence-to-sequence model

Our baseline model is a standard encoder-decoder model that uses LSTM cells for both encoding and decoding, albeit with different weights for encoding and decoding. In the encoder, a token of the article x_i are fed at each time-step t. All embeddings were initialized from a random uniform distribution and learned during training. Any sequence longer than the max encoding length was truncated to fit the encoder, and short sequences were truncated to fit. For the decoder, all reference headlines were prefixed with the START, $< s >$, token and suffixed with the END , $< /s >$, token.

Attention model

In the attention model, at every decoder state we generate a context vector based on all the hidden states generated in the encoding stage. The context vector at a given decode stage i can be computed with encoding hidden states h_j and the previous decoder hidden state s_{i-1} with a weight

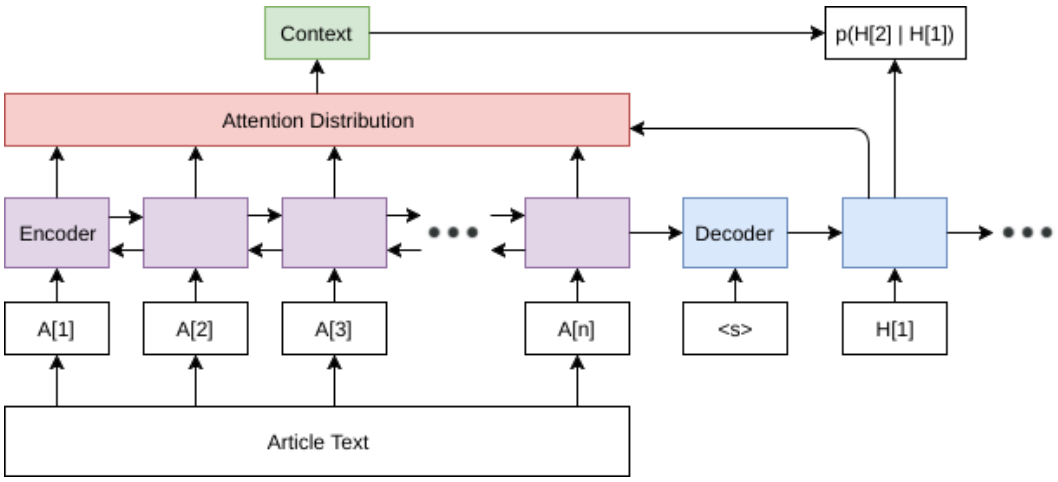

Figure 2: bidirectional encoder sequence to sequence model with attention

matrix W_a having dimensions $D_h \times D_h$,

$$a_j = \text{softmax}\left(h_j^T W_a s_{i-1}\right) \tag{3}$$

$$c_i = \sum_{j=1}^{N} a_j h_j \tag{4}$$

In the above equation, N is the encoding length used. From the context vector, the previous hidden state, and the input to the current decoder LSTM, the new output word distribution can be generated. The algorithm used for this was derived from previous work on attention based mechanisms [2]. The algorithm is described as follows:

$$\tilde{t_i} = U_o s_{i-1} + V_o E y_{i-1} + C_o c_i \tag{5}$$

$$t_i = \left[\max\{\tilde{t}_{i2j-1}, \tilde{t}_{i2j}\}_{j=1,\ldots,l}^{T}\right] \tag{6}$$

$$p(y_i|s_i, y_{i-1}, c_i) \propto \exp\left(y_i^T W_o t_i\right) \tag{7}$$

In this case, E is the Embedding matrix and y_i is a one-hot vector, such that Ey_{i-1} is the word embedding for the $(i-1)^{th}$ word.

Bi-directional model

In the bi-directional model, the encoder uses a bi-directional LSTM, which concatenates the hidden states at each time step from both the previous and next word, capturing semantics from both sides of each word. The number of hidden units in each encoder LSTM was half of the number of hidden units in the output, so that in the same number of units the forward and backward state can be concatenated. Although this would not provide any meaningful benefit for a simple sequence to sequence model, it would provide a much better attention insight due to the attention model being able to see the past and future of each given word in the input text.

Pointer-generator model

In the pointer model, we take into account words that may not appear in our vocabulary, but may be essential to generating the title. Examples of that could include names of people/places or numbers. In order to account for this, a pointer generator model was constructed based on some previous and current work [8]. In this model, both the distribution generated from the context and the decoder output vector and the distribution of scores generated for the attention is used together, with a p_{gen} factor to combine them together. the probability of generation p_{gen} is calculated as follows:

$$p_{gen} = \sigma\left(W_{h^*}^{'} h_t^* + W_s^{'} s_t\right) \tag{8}$$

The sigmoid is used because this determines the probability of whether to point to the original input or use a token from the vocab. From this, the probability distribution of the final output can be

computed as follows:

$$P(w) = p_{gen}P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t \quad (9)$$

This allows us to both find the probability distribution over all the words in the vocabulary, as well as over all the words in the input text which may not be in the vocabulary. This model allows the decoder to check if an unknown word could be essential when generating a summary for the article. The model can be seen in Fig. 3.

Dataset

For all of our experiments, we used the Gigaword Corpus [1]. We created our own scripts to preprocess to the data to output the first three sentences of each article along with its corresponding reference headline, resulting in roughly 1.2M article-headline pairs. We used 10%, 120K examples each, for our development and test sets. We took the top 100K most-common words in our dataset as our vocab . Each non-ascii character was assigned its own unique token and all numbers were replaced by an $< NUM >$ token.

Experiments

Similar to Google's textsum summarization model, we use a 128-dimensional word embeddings and 256-dimensional hidden states for our LSTM cell. Word embeddings were learned through training. For efficient training, we used a batch size of 64. For our optimizer, we use Adam with a learning rate of 0.01 and $b_1 = 0.9, b_2 = 0.99$ as its parameters. We train for 10 epochs. At each epoch, the development loss was calculated and only the model with the lowest development loss was saved.

To help prevent overfitting, during training, we use dropout with a rate of 0.75 (p_{keep}) at each time step.

To determine our lengths for the encoder and decoder, we plot the sentence length distribution for all articles and headlines.

Based on our plots, we used an max encoder length of 200 and max decoder length of 20.

Generating Summaries

At test time, to generate our summaries, we want to output a sentence, $\mathbf{y}^*$ that maximizes the sum of the log probabilities of each word in the decoder.

$$y^* = \arg\max_{y \in Y} \sum_{i=0}^{N-1} \log p(y_{i+1}, \mathbf{x}) \quad (10)$$

where $\mathbf{x}$ is the input article.

As used commonly in sequence-to-sequence models and considered a standard approach for neural MT models, we approximate the arg max of the function with two methods: greedy and beam-search.

In greedy, the arg max of the softmax distribution at each current time step is used as the input to the next time step of the decoder.

In beam-search, we improve our search for a local optimum by calculating the best K potential hypothesis at each time step of the summary. Then, each of those hypothesis is fed at the next time step, resulting in K^2 potential hypothesis, and again, only the top K hypothesis are kept. The algorithm is outlined in Algorithm 1.

In the algorithm, y is the list of indices of tokens, *states* is the list of hidden states of the LSTM, and *probs* are the softmax log probabilities of each token in the vocabulary.

Results

We calculate the cross-entropy loss from each of our models from the train, dev, and test set. Our results are given in table [1]. Because our pointer-generator model was too large, we could not train the model with vocab size of 100k due to memory constraints. Therefore, we re-ran our models with vocab sizes of 15k to fairly compare the results.

The results show a disparity between train loss and the dev/test loss. This is because in the training stage, the loss was computed by the probability of the actual next word given the correct previous word. The dev/test loss was computed by having the model generate a greedy search output, and finding the probability of the actual labels given the greedy distributions.

In addition, it appears that a 15k vocabulary for a simple attention model is the best model

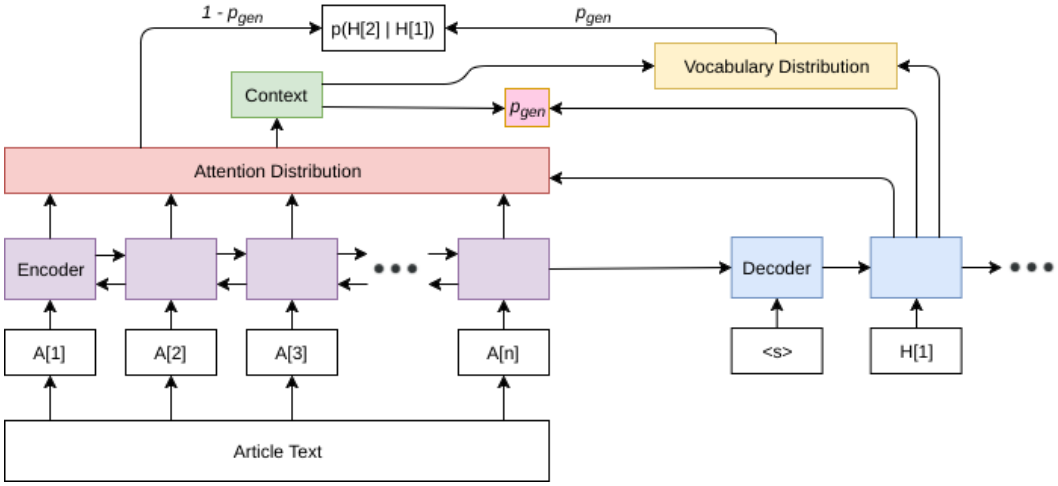

Figure 3: sequence to sequence model bidirectional encoder, attention, and pointer. At each decode timestep, the probability $p_{gen} \in [0, 1]$ is computed based on the current decoder hidden state and the context vector. It is used to determine the final vocabulary distribution.

Algorithm 1: Beam Search(x)

```
# x is output from the encoder
N ← max decode length
start ← < s >
initial_state ← zero_state
for t = 0  to  N − 1 do
    if t = 0 then
        y, states, probs ← output(start,
          initial_state, x)
        y_t, state_t, p_t ← topK(y, states,
          probs)
    else
        all_states ← {}, all_y ← {}, all_probs
          ← {}
        for i=1 to k do
            y, states, probs ←
              output(y_{t−1}[i], state_{t−1}[i], x)
            y_t, state_t, p_t ← topK(y_{t−1}[i]+
              y, states, p_{t−1}[i] + probs)
            all_y ← all_y ∪ y_t
            all_states ← all_states ∪ state_t
            all_probs ← all_probs ∪ p_t
        y_t, state_t, p_t ← topK(all_t,
          all_states, all_probs)
y ← argmax_y(y_{N−1}, p_{N−1})
return y
```

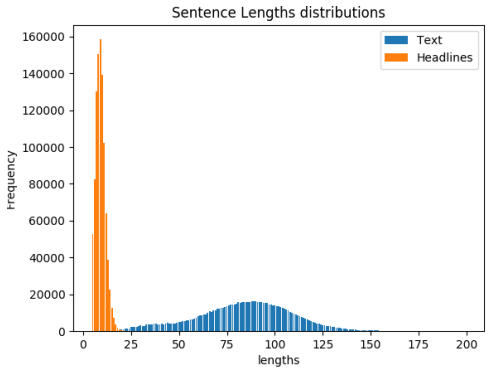

Figure 4: Article length distribution

for the summarization task. However, it was noted that the learning rate and number of epochs wasn't optimized for each model to reach convergence. Due to this, models with less learn-able parameters are closer to reaching convergence values and therefore perform better. Given the amount of time used for training and testing, it was not possible to train each model to convergence and accurately show how well they would perform after many more training epochs.

Qualitative Results

To observe qualitatively from our models, we used beam-search to produce the summaries. In the following tables, we illustrate some examples that

worked very successfully. Because the model also learns the grammatical structure of the english language, our generated summaries were often more grammatically correct than the actual reference headline. It is important to note, however, that most of the words from the generated summaries come from the first sentence of the source text and less often includes words that appear in later sentences.

	Train Loss	Dev Loss	Test Loss
Basic	2.9944	4.3660	4.3649
Attention	1.5384	3.6221	3.6091
Bidirectional	1.9091	3.5815	3.5947

Table 1: Cross-entropy loss of each model with vocab size of 100k. Each model is built on top of the previous model.

	Train Loss	Dev Loss	Test Loss
Basic	2.3578	5.5763	5.5605
Attention	1.3661	3.2652	3.2543
Bidirectional	1.3829	3.3253	3.3157
Pointer	1.4898	3.4649	3.4560

Table 2: Cross-entropy loss of each model with vocab size of 15k.

Original text (first 3 sentences) dollar trades at upper 96 yen level in tokyo tokyo , nov . 19 (xinhua) – the usa dollar traded within the upper 96 yen range wednesday in tokyo . at 5 pm.
Reference Summary dollar trades at upper 96 yen level in tokyo
Basic Seq2Seq china 's $< UNK >$ to be held in
Basic Seq2Seq + Attention dollar trades at upper $< NUM >$ yen level in tokyo
Basic Seq2Seq + Attention + Bidirectional dollar trades at upper $< NUM >$ yen level in tokyo
Pointer-Generator dollar trades at upper 96 yen level in tokyo

Table 3: Results showing how the pointer is able to extract numbers that it has never seen before such as "96".

Original text (first 3 sentences) china rejected thursday a demand from pope john paul ii for religious freedom and warned the vatican to stop interfering in china 's domestic affairs . the vatican " must cease its interference in china 's internal affairs , including interference in china 's affairs by making use of religion , " a foreign ministry spokesman said . " the chinese government respects the freedom of religious belief of chinese citizens , and protects their regular religious activities , " he added .
Reference Summary china rejects pope 's demands
Basic Seq2Seq china 's <UNK >to be held in
Basic Seq2Seq + Attention china rejects pope for religious freedom
Basic Seq2Seq + Attention + Bidirectional china rejects vatican demand for religious freedom
Pointer-Generator china rejects call from pope for religious freedom

Table 4: Results showing how different models are able to capture the same summary using different wording and may even be better than the reference summary. This particular result shows pointer model being more grammatically correct than the others.

Original text (first 3 sentences) russian energy exports to the european union are expected to increase in coming years , the industry and energy minister said wednesday . russian news agencies quoted viktor khristenko as saying that diversifying export routes and the development of new oil and gas fields would help fuel the increase . the reports did not say by how much the exports were expected to increase .
Reference Summary russian energy exports to eu expected to increase , minister says
Basic Seq2Seq + Attention Greedy Search russian energy exports to eu to increase in coming $< UNK >$
Basic Seq2Seq + Attention Beam Search russian energy exports expected to increase in coming years

Table 5: Results showing how the model can better generate meaningful summaries when using the beam search

Conclusion

In this paper, we presented a neural sequence-to-sequence model with attention mechanism as well pointer-generator model. Our models produced very promising results and the pointer-generator was effective in pointing back to source text key words for out-of-vocabulary words.

However, we believe if trained with larger datasets and for more epochs, our models could reach their full potential in abstractive summarization. In our future work, we hope to optimize our various training hyperparameters. Additionally, because articles are usually longer than three sentences, designing architectures that can handle longer source text such as the *CNN/Daily Mail* dataset as well as learning how to effectively capture content that appear much later in the article would be the next logical steps to take.

References

[1] David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2003. English gigaword. Linguistic Data Consortium, Philadelphia.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473.

[3] Greg Durrett, Taylor Berg-Kirkpatrick, and Dan Klein. "Learning-Based Single-Document Summarization with Compression and Anaphoricity Constraints" ACL 2016.

[4] Ke Zhang, Wei-Lun Chao, Fei Sha, Kristen Grauman. "Video Summarization with Long Short-term Memory" ECCV 2016.

[5] Lin, Chin-Yew. 2004. ROUGE: a Package for Automatic Evaluation of Summaries. In Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain, July 25 - 26, 2004.

[6] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. 2016. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. In Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL 2016)

[7] Rush, A.M., Chopra, S., and Weston, J. A neural attention model for abstractive sentence summarization. In EMNLP, 2015.

[8] See, Abigail, Get To The Point: Summarization with Pointer-Generator Networks, under review.